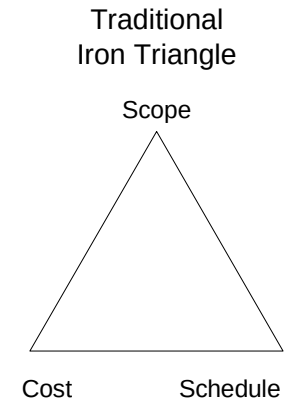
The Quality Dial

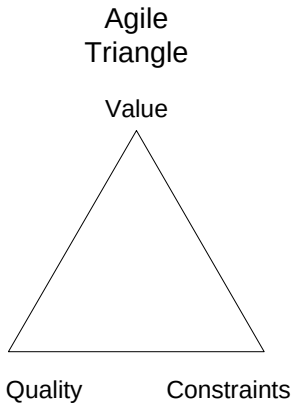
Joshua Kerievsky
joshua@industriallogic.com
Twitter: JoshuaKerievsky
QCon San Francisco
November 2009

The Quality Dial

Adapting Over Conforming



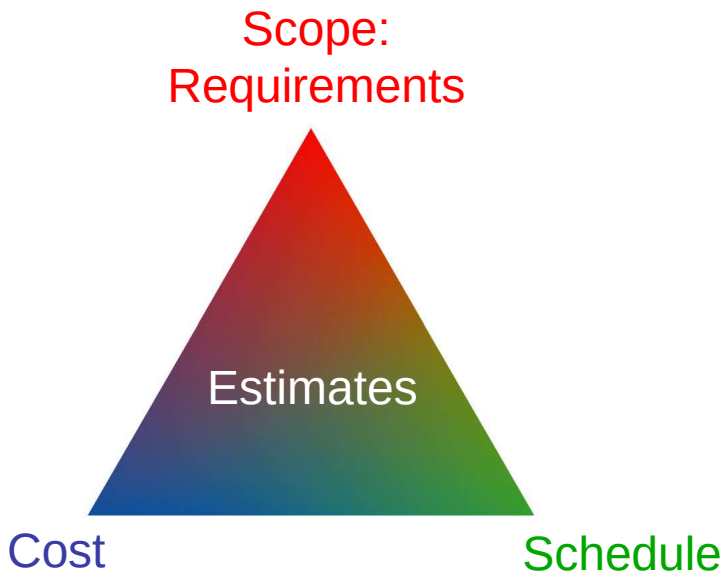
- Rewards conforming
- Little trust
- Constraint oriented
- Focus on control



- Rewards adapting
- Significant trust
- Value & quality oriented
- Focus on delivering results

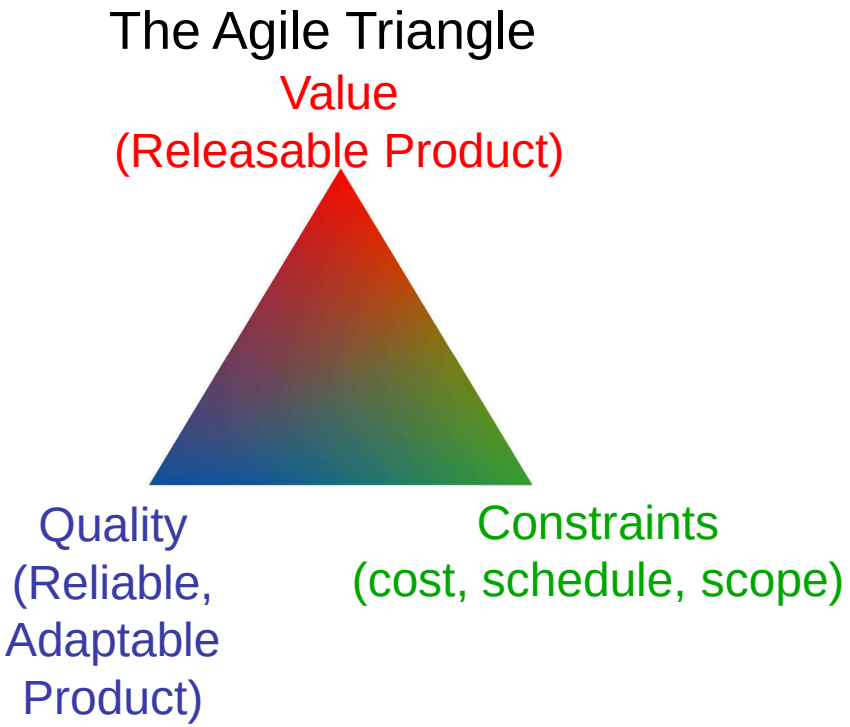
Traditional Development

- Project focus (benefits)
  - Short term
  - Deploy at the end
- Requirements are fixed, plan is fixed and inflexible
- Performance: schedule, cost, scope
- Actions: resist change, cut testing, cut quality

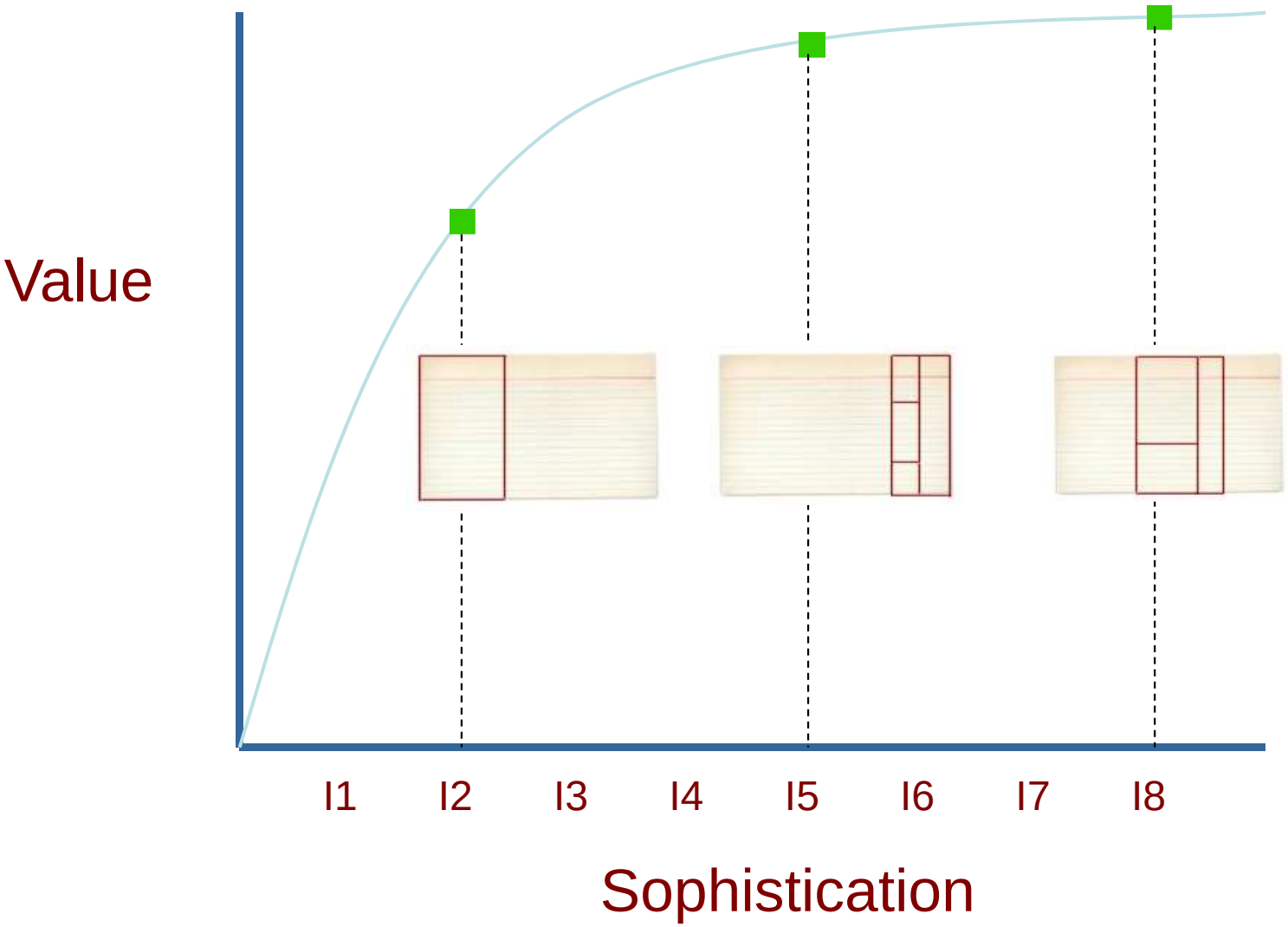


Agile Development

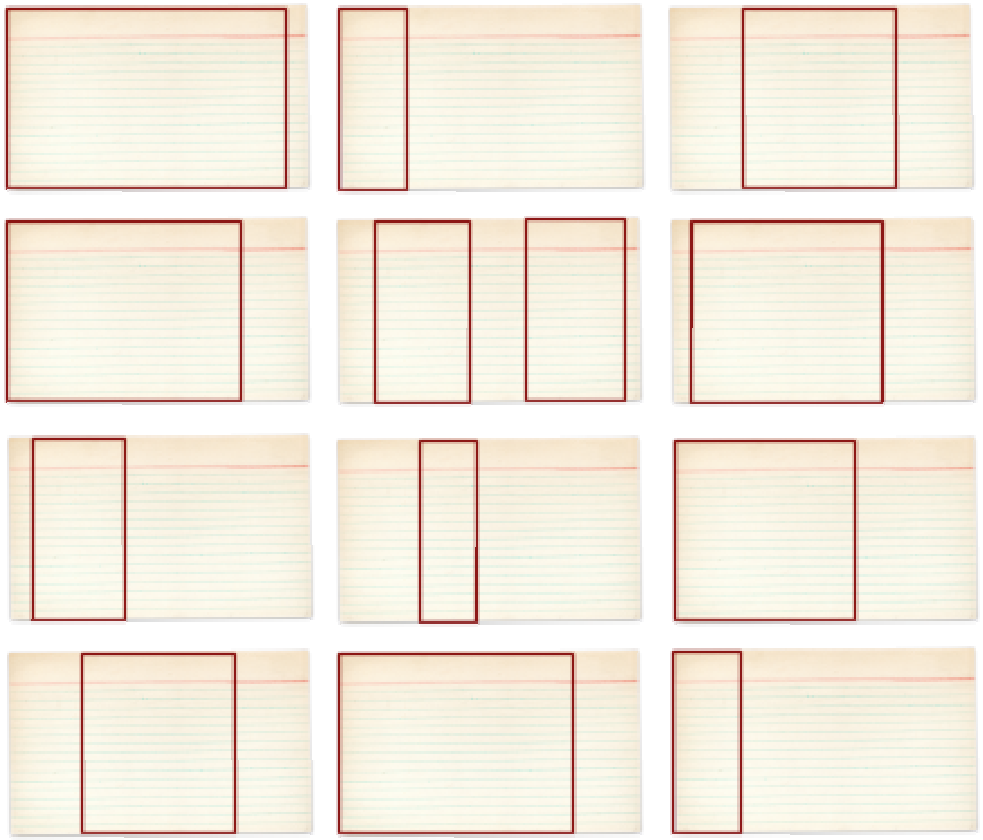
- Product focus (benefits)
  - Both short & long term value
  - Continuous deployment
- Goals drive, plans guide, adaptation expected
- Performance: value, quality, constraints
- Actions: adapt to change, adjust features, resist quality reduction

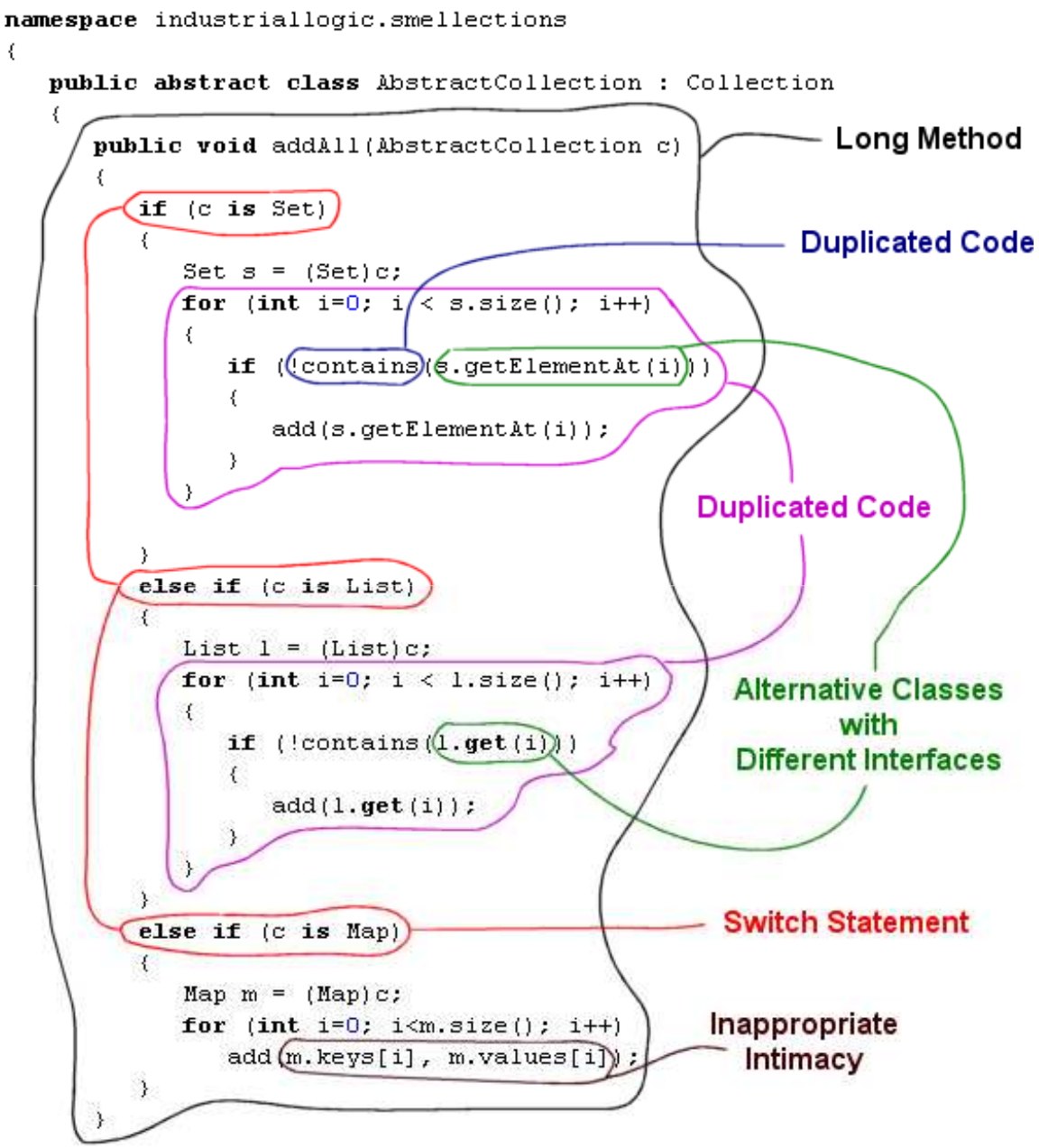


Evolution of a Feature



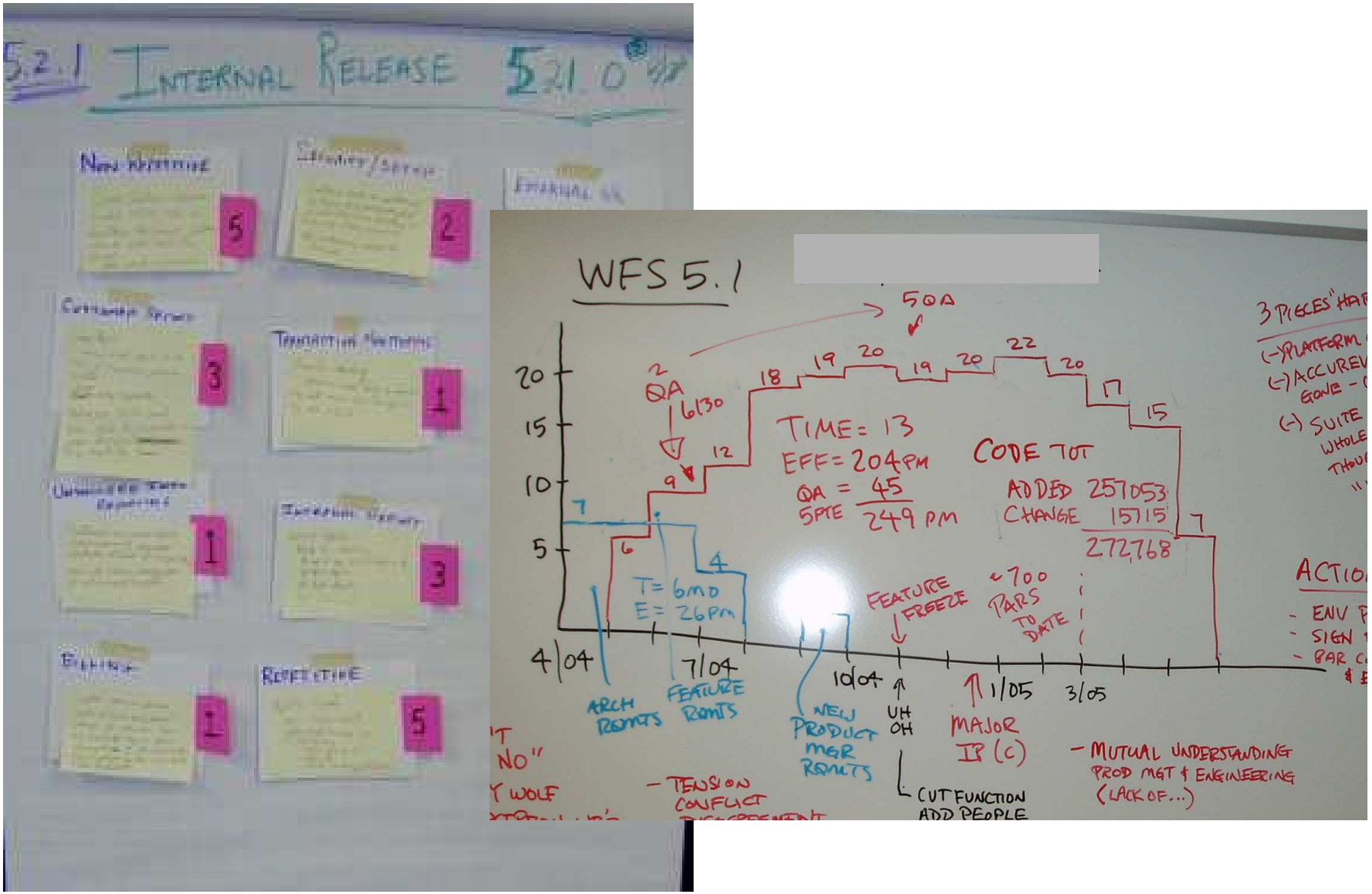
The Art of Release Planning




namespace industriallogic.smellections
{
public abstract class AbstractCollection : Collection
{
public void addAll(AbstractCollection c)
{
if (c is Set)
{
Set s = (Set)c;
for (int i=0; i < s.size(); i++)
{
if (!contains(s.getElementAt(i)))
{
add(s.getElementAt(i));
}
}
}
else if (c is List)
{
List l = (List)c;
for (int i=0; i < l.size(); i++)
{
if (!contains(l.get(i)))
{
add(l.get(i));
}
}
}
else if (c is Map)
{
Map m = (Map)c;
for (int i=0; i<m.size(); i++)
add(m.keys[i], m.values[i]);
}
}
Long Method
Duplicated Code
Duplicated Code
Alternative Classes with Different Interfaces
Switch Statement
Inappropriate Intimacy

CO^2

5.X Software


5.2.1 INTERNAL RELEASE 5.2.0
WFS 5.1
TIME = 13
EFF = 204PM
QA = 45
SPTE 249 PM
CODE TOT
ADDED 257053
CHANGE 15715
272768
T= 6MO
E= 26PM
FEATURE FREEZE
~700 PAIRS TO DATE
4/04
7/04
10/04
1/05
3/05
ARCH RQMTS
FEATURE RQMTS
NEW PRODUCT MGR RQMTS
UH OH
MAJOR IP (C)
CUT FUNCTION ADD PEOPLE
- MUTUAL UNDERSTANDING PROD MGT & ENGINEERING (LACK OF...)
- TENSION CONFLICT
3 PIECES

Only Refactor When Working On A User Story?

- Common Agile wisdom suggest that we should only refactor code and improve quality in conjunction with work on a User Story.
- This ensures that we are working on Customer-valued work instead of work that only Developers value.

Refactoring Pause

- There are times when you simply need to pay down technical debt. You need a Refactoring Pause.
- We have simply incurred more technical debt than we would like and now is a good time to pay it down.
- There has been a pernicious Singleton that has played a central role in our code and is now being killed off, because that will open up our code for many more design improvements.

Why Pay Now?

- Why was it better to do this now rather than to defer each change until you were passing through that part of the code? –Ron Jeffries
- Some smells sprawl themselves across a design. They are not easy to fix, otherwise they'd have been fixed a long time ago. Our Singleton friend is one such smell. Without a dedicated effort from 2-3 of us for several days, the Singleton would still be as strong and pernicious as ever. Today, it is in a hospice. -Joshua Kerievsky

Knowingly Incurring Design Debt? Huh?

- You are doing TDD and continuous refactoring, and nonetheless you claim that you "knowingly incurred some technical debt." How? Why? Please explain. Other than the fact that it is pernicious how is this technical debt so onerous that you choose not to deal with it when it emerges? - Adam Sroka

Pernicious & Easy Design Problems

- We needed Playlists in our product. Got them working in 2 weeks. We TDDed and refactored along the way, however, we were not *merciless* in refactoring due to the deadline. For example, our UserLibrary class got too fat with responsibilities. Hard to fix? Not at all. It is not a pernicious design problem, just an annoyance. On the other hand, our Singleton is pernicious. We largely ignored it and could've kept on ignoring it, yet most of us identify that Singleton as a foul smell. -Joshua Kerievsky

Sacrifice Quality for Speed?

- But, you felt like you had to rush and ignore some of your better instincts to do it. That smacks of an "epic" to me. Wouldn't it have been better to break the playlists up into smaller chunks each of which included "merciless" refactoring rather than to end up throwing out quality in favor of completing the epic inside the iteration? - Adam Sroka

Delivery Now, Mercilessly Refactor Later

- The key was to design and make playlists a reality in time for a deployment to our client. We did that in little, test-driven steps, releasing along the way. Cutting a few corners on some design was an easy decision, especially since we planned to mercilessly refactor post release. **This is simply good business practice.** Blowing a deadline because we want the design to be perfect would've been bad business.

– Joshua Kerievsky

Taking On & Paying Down Debt

- When you run a business, you make decisions about taking on debt and paying down debt and you do it all the time. Problems come when you ignore your debt for too long. I think we walk a healthy line there. We pay debts all of the time and we let a few debts accumulate until we get a good chance to pay them off. It's all about finding the right balance to remain competitive. - Joshua Kerievsky

Dangerous Advice?
For Experts Only?

- I fear that the advice to perform technical work outside of a User Story is very dangerous in the context of Scrum and XP. Particularly for teams that haven't quite understood delivering constant, incremental business value in a customer friendly way. - Adam Sroka

The Real Danger?
Not Paying Down Design Debt

- There is no one right way to pay off technical debt. We like to do it continuously in the context of user stories. Yet we also find that occasionally we need to stop and attack some pernicious design issue. If we did that without approval from the business, we'd be violating a core community agreement. So it must be sanctioned and it must be important enough that it is as valuable to do as working on the next feature. -Joshua Kerievsky

A Puzzlement

- Josh's experience is almost completely inapplicable to any team without those same characteristics, especially a customer who understands the need for refactoring very deeply. Understanding that need very deeply is risky too. Someone like me, who really gets how important good code is, may go too far and prefer sweet code when another feature would have made for more success. Once we get behind, we're in a very risky spot. Yet we WILL get behind. It's a puzzlement. – Ron Jeffries

A Driving Metaphor

Good software management is a dance between adding features & maintaining quality. If the former trumps the latter too much, the product/company/staff suffer. If the latter trumps the former too much, the product/company/staff suffer. In an ideal world, we strike a balance. In the real world, we
* push the pedal all-the-way-down now,
* ease up a little over there,
* now hit the breaks,
* now slowly accelerate. - Joshua Kerievsky

New Agile Teams & Debt

- New Agile teams incur tons of technical debt in their first few iterations. They haven't learned to refactor continuously or lack good design skills or they are new to a framework and aren't doing things the right way. Within a very short period of time (e.g. 2-3 weeks), they can produce a smelly, unstable foundation on which they should *not keep building*. It's better to stop, fix the mess, then move on. And that must be a business decision -- it can't just be the developers deciding to do it. - Joshua Kerievsky

Refactoring Over Features?

- Why does the customer value debt removal more than new features? Does this suggest that the customer is running low on good feature ideas? - Ron Jeffries
- Sometimes cleaning up really bad smells is more valuable to an organizationthan adding new features. Not always true, yet if you ignore those bad smells for too long, and always push features, you can get yourself into a world of hurt. - Joshua Kerievsky

Faster Feature Injection

- Some smells sprawl themselves out across a design. They are not easy to fix, otherwise they'd have been fixed a long time ago. Our Singleton friend is one such smell. Without a dedicated effort from 2-3 of us for several days, the Singleton would still be as strong and pernicious as ever. Today, it is in a hospice. -JoshuaKerievsky

Clean Code Improves Velocity

- To me, at least, keeping the code base sufficiently clean that you can maintain (or improve) velocity is a primary business goal. -John Roth

Cleaning Up Along The Way

- Too many companies ignore the increasing complexity of their code, poor quality, etc. It's great to be competitive and get code out the door, yet for the long haul, it's wise to clean up along the way. - JoshuaKerievsky

Future Payoff?

When we clean up the code, the cleanup we make will only pay off at some future time. Some may pay off tomorrow, and some may not pay off for weeks or months. None of it pays off now. All refactoring that slows feature progress is an investment in the future. What needs to be figured out is whether, how, and when, such an investment is really worth making. - Ron Jeffries

More Features
Not Always Good!

- I'm not even convinced that features add value. Ron seems to assume that. I think most products would do well to have more analysis about what exact features are really needed and will be valued by customers rather than the incessant injection of speculative features into a system and the notion that constant feature injection always equals "good forward motion." - Joshua Kerievsky

Looping Forever On Bad Design

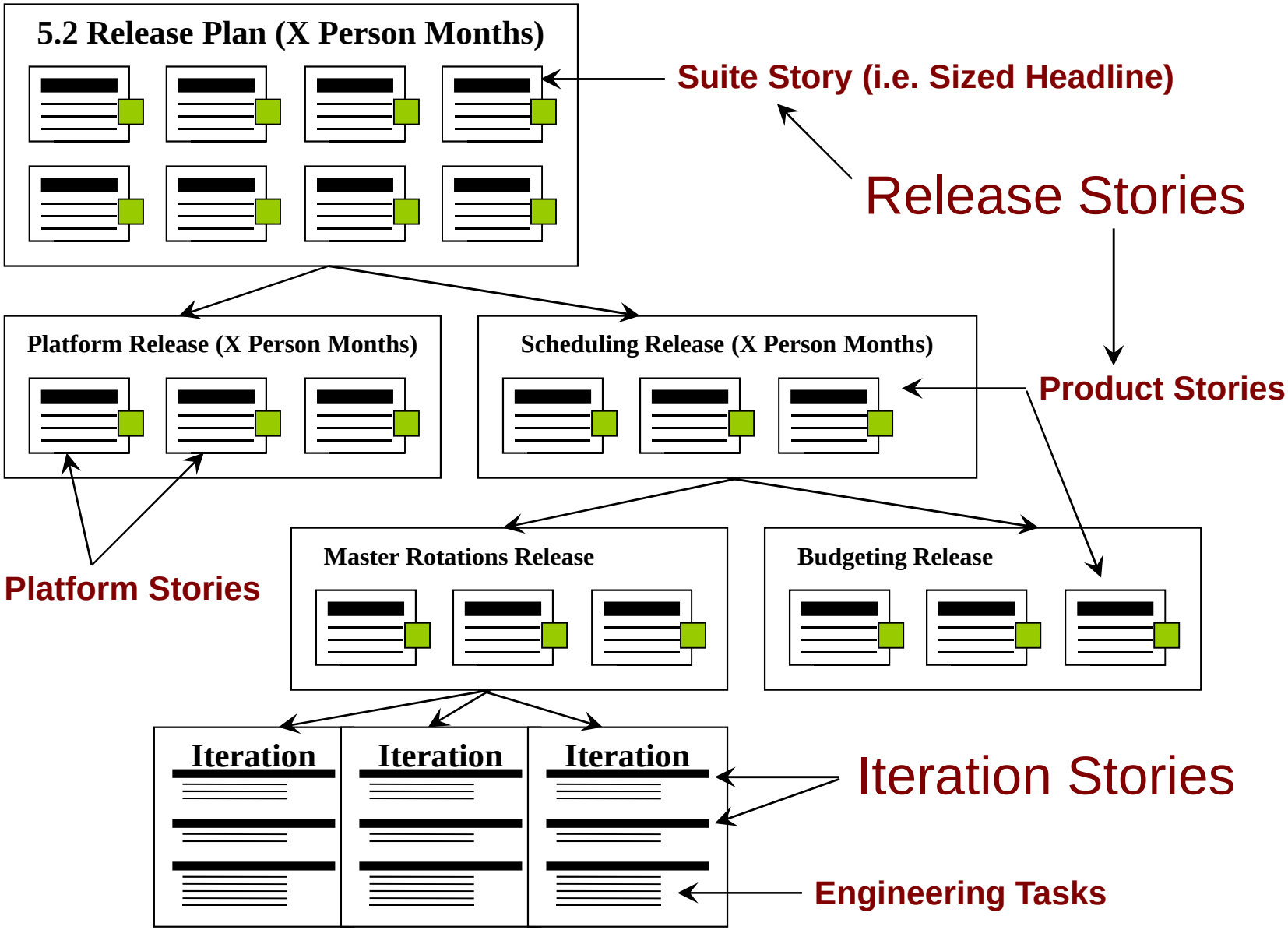
- It is entirely possible to stop, refactor, be a little behind on features, slam in a few features, not catch up but screw up the code, and loop forever, never getting benefit. We hope that's unlikely ... and it is, if people are sufficiently skilled ... which is part of my point above, that your advice is good for experts. - Ron Jeffries

Business Context!

- The above makes no recognition of business context. Say a company has been slamming in features 2 months for some important delivery, doing modest refactoring all along the way, yet not enough to keep the design very clean. If that company stops to refactor, they are NOT A LITTLE BEHIND ON FEATURES, as you say. They are ahead. - Joshua Kerievsky

Refactoring Pause
An Inferior Decision

- Yes, stopping to refactor is a business decision. In my opinion, it is almost always an inferior one. Also, in my opinion, we almost never have enough information to evaluate the decision on a business basis. To do so, we would have to know the value creation curves with and without the pause. - Ron Jeffries


5.2 Release Plan (X Person Months)
Suite Story (i.e. Sized Headline)
Release Stories
Platform Release (X Person Months)
Scheduling Release (X Person Months)
Product Stories
Platform Stories
Master Rotations Release
Budgeting Release
Iteration
Iteration
Iteration
Iteration Stories
Engineering Tasks